5. Sub-typing, Interfaces, Abstract Classes & Inheritance

M2104 Conception Objet

Édition 2021

Sous-typage et polymorphisme

- Les **types** définissent les valeurs et comportement des individus (dont variables, attributs… objets compris)

- Les types sont organisés en **hiérarchie** (les sets sont des collections…)

- Créer des classes qui sont sous-types d'un même type permet d'avoir du **polymorphisme** : la possibilité d'exécuter de manières différentes une même opération (comme / sur les nombres à virgule flottante ou les nombres entiers)

- Cela permet de **réduire les redondances** en plaçant le code dupliqué à un seul endroit ; d'avoir **différents comportements** pour différents sous-types, et facilite l'**abstraction**

Abstraction

- Abstraction : **dépendre de la signature d'une méthode et non de son code**
- En détail, le code précis d'une méthode dans type **général** n'est pas défini par ce type mais par un type plus **spécifique**
- Les classes utilisant le type général ne dépendent donc que des **paramètres**, du **type de retour**, du **nom de la méthode** et de la documentation
- La signature de la méthode est la **déclaration** de la méthode ; sans implémentation, c'est une **méthode abstraite**
- Une méthode abstraite est une **obligation contractuelle** à tout sous-type de donner une implémentation (du code) pour cette méthode ou de **déléguer** ceci à d'autres classes

Interfaces

- **Une interface** est une collection de **déclarations de méthodes** (signatures, méthodes abstraites) constituant un **contrat** déterminant **ce que les objets de ce type peuvent faire**

- Les interfaces peuvent aussi avoir des éléments ‹‹ d'interface ›› (static) et par défaut

- Les interfaces doivent être **réalisées** / **implémentées** par des classes respectant le contrat en **donnant du code** : les définitions de méthode

- En Java, une interface qui ne déclare qu'une méthode est une **interface fonctionnelle**

Classes abstraites

- Les **classes abstraites** contiennent les mêmes éléments que les autres classes, dont attributs, méthodes, constructeurs… mais **ne peuvent pas être instanciées directement**

- Elles peuvent également contenir des **méthodes abstraites** - des déclarations de méthodes, comme les interfaces (classes et méthodes sont notées *abstract* en Java)

- Comme pour les interfaces, les classes abstraites définissent des **contrats** pour les classes qui en **héritent** (les méthodes abstraites doivent être **définies**, donc implémentées par du code)

- Les classes abstraites sont couramment utilisées pour **réduire des redondances de code**

Héritage général

- Les classes peuvent **hériter** d'autres classes (pas seulement de classes abstraites) ; les interfaces peuvent aussi hériter d'autres interfaces

- Les classes du même paquet héritent des attributs et méthodes **restreints au paquet**, les classes dans les autres paquets héritent des méthodes et attributs publics et **protégés**

- La visibilité **protégée** (#) est utilisée pour cela (en Java)

- Les méthodes et attributs **privés** ne sont jamais accessibles dans les sous-classes

- Les sous-classes **peuvent** re-définir (*override*) les méthodes dont elles héritent

- Toutes les classes **concrètes** (les classes pouvant être instanciées) **doivent** définir toutes les méthodes abstraites des classes / interfaces dont elles sont des sous-types

Un exemple d'organisation complexe